ENABLING PARTIALLY RECONFIGURABLE IP CORES PARAMETERISATION AND INTEGRATION USING MARTE AND IP-XACT

G. Ochoa-Ruiz, O. Labbani, E. Bourennane
LE2I Laboratory, Université de Bourgogne, France
C. Author: gilberto_ochoa-ruiz@etu.u-bourgogne.fr

S. Cherif, S. Meftali, J.-L. Dekeyser
INRIA Lille Nord Europe, Villeneuve d'Ascq, France
(FirstName.SecondName)@inria.fr

Abstract – **This paper presents a framework which facilitates the parameterization and integration of IP cores into partially reconfigurable SoC platforms, departing from a high-level of abstraction. The approach is based in a Model-Driven Engineering (MDE) methodology, which exploits two widely used standards for Systems-on-Chip specification, MARTE and IP-XACT. The presented work deals with the deployment level of the MDE approach, in which the abstract components of the platform are first linked to the lower level IP-XACT counterparts. At this phase, information for parameterization and integration is readily available, and a synthesizable model can be obtained from the generated IP-XACT through model transformations. We detail how certain IP-XACT objects are exploited in our approach; the emphasis is given to the generation of IP cores in a Xilinx EDK environment. We provide a case study in which a complete DPR platform is modeled in MARTE and implemented in a FPGA.**

Index Terms—Dynamic Partial Reconfiguration, UML MARTE, MDE, IP-XACT, EDA.

1. INTRODUCTION

Dynamic Partial Reconfiguration (DPR) [1] enables a set of Partial Reconfigurable Modules (PRMs) to time-share a pre-defined portion of the programmable fabric while the remainder of the logic in the system remains active. A DPR platform has advantages over conventional FPGA systems, including partial updateability, increased flexibility, reduced resource utilization, and low power dissipation [2]. Despite its benefits, the DPR implementation process remains complex and time consuming, and often requires hardware developers to have a thorough understanding of the underlying fabric and design methodology.

Furthermore, the creation of SoC DPR-based systems requires IP reuse capabilities in which the parameterisation and integration, both of DPR and non-DPR components, is performed in such a way that facilitates the design process.

Model Driven Engineering (MDE) in tandem with UML has been used in co-design methodologies in the last years with relatively success in embedded systems modeling [3]. Many of them made use of the UML profile for "Modelling and Analysis of Real Time and Embedded Systems" (MARTE) [4].

UML/MARTE models are used not only for communication purposes but, using model transformations, to produce concrete results such as a source code. For this purpose, MDE methodologies make use of a deployment phase in which the building blocks of the high-level models are linked to the low implementations that embody the related behaviour. This is basically an IP reuse problem, and in this way the components can be configured, and a synthesizable top-level implementation can be obtained.

In this paper we proposed an MDE component based approach for the parameterization and integration of IPs into DPR SoC based systems. The generation framework departs from a deployment model in MARTE that is obtained from high-level library, containing abstract representations IP-XACT objects [5]. The parameterisation and integration of the platform takes place at this level, effectively abstracting the low-level implementation details.

The components in the MARTE model are linked to IP-XACT representations that similarly reference the low-level implementation files, used for generating the EDK system.

This description is subsequently synthesized, and along with the DPR IP netlists, is fed to the PlanAhead DPR design flow. In this way, we provide framework for facilitating the design entry phase of the DPR design flow.

The rest of this paper is organized as follows: in Section II, we describe briefly the similar approaches and its limitations regarding the deployment phase. Then, in Section III, we introduce the proposed methodology, focusing on the deployment phase onwards. In Section IV we embark in a thorough discussion in which we detail how certain IP-XACT concepts are used in our framework. In section V, a case study for the creation of EDK-based DPR system is presented. Finally, we conclude and provide some avenues for future work.

2. RELATED WORKS

Several works have tackled the use of MARTE in SoC design, specifically at the deployment level, such as the MoPCoM [6] and GASPARD [7] frameworks. The main disadvantage is that, as with many other MDE methodologies, both approaches make use of non-

standardized deployment representations. This means that the deployment models obtained by these methodologies are not interchangeable, making them highly methodology-dependant. Another issue is that the models need to be annotated manually, and the parameters are retrieved from the implementation files in a non-automatic way.

This problem has been addressed by the SoC industry, by developing the IP-XACT standard, which aims at facilitating the configuration, integration, and verification in multi-vendor SoC design flows though a set of structured XML schemas. Several industrial cases studies have demonstrated that the adoption of IP-XACT facilitates the configuration, integration, and verification in multi-vendor SoC and IP integrating design flows [6], [7].

The standard has risen the interest of the academia to bridge the gap between high-level approaches and IP-XACT [8, 9], where the latter is used as an intermmediate representation to perform a series of tasks, but varying in the intended back-ends (e.g. RTL, SystemC). However, these approaches do not present a means of "importing" IP important parameters from the low-level implementations files to facilitate their deployment.

The contributions of this paper are the introduction of an MDE approach that uses the UML MARTE profile, and that enables moving from high level models to HDL code generation. IP-XACT is used as an intermediate model, used to configure the deployed IPs in the platform and to automate the integration of the DPR and non-DPR IPs. The parameters needed at high levels of abstraction are stored in the deployment level, facilitating the task of the designer. The parameterised system and IPs are then used to generate the necessary inputs to the DPR design flow. Our approach simplifies the conception of FPGA-based SoCs, facilitates the composition and generation of DPR designs.

3. PROPOSED METHODOLOGY

In this section, we embark in a thorough description of the proposed framework for DPR IP integration and parameterization, emphasizing how it is embedded into the design flow of DPR systems. The MDE methodology is based on a Y schema approach. As with other co-design approaches, the system specification starts by modeling the application and architecture separately, which are subsequently associated and deployed at a high-level of abstraction. The models obtained at the deployment phase are use for code generation, and we provide means for obtaining the pertinent parameters, options and interfaces of the IP blocks, thus facilitating the designer's task.

3.1. Deployment phase of the MDE methodology.

.

The deployment phase of any MDE methodology is instrumental, since enables the generation of a synthesizable SoC description from a high-level MARTE model. More precisely, sufficient information must be provided at this stage so that the code integration and parameterization on the IPs can be performed.

The proposed framework, in terms of models transformations and departing from the deployment phase, is depicted in Figure 1; the green line describes the actual tool flow**.** It uses three levels of abstraction, each making use of its corresponding component library. The entry point is a MARTE deployment platform model (a Composite Structure Diagram, CSD), which is created by choosing components from a Component Model Library (CML). The MARTE model is to be obtained after the association phase, where sufficient information about the components to use is available. At this phase, components are seen as simple IP blocks containing interfaces to be connected and parameters to be set by the designer.

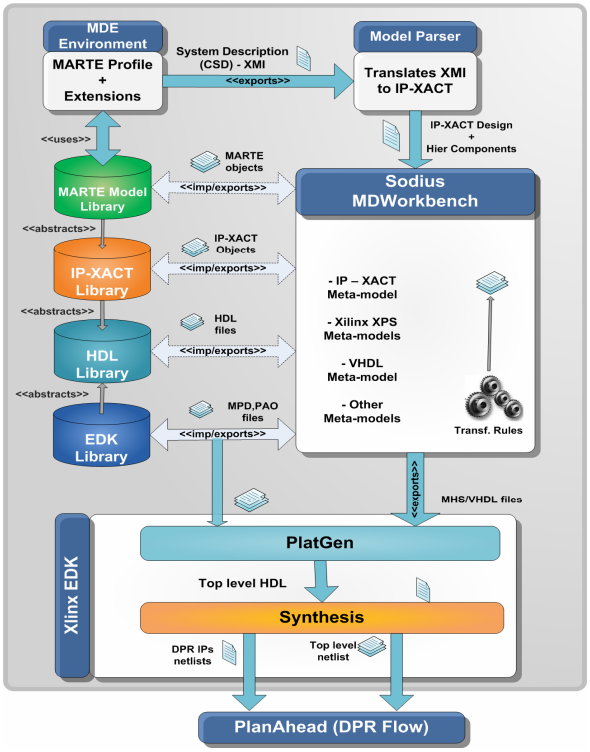

Fig 1. MDE framework for DPR system and IP composition

3.2. Models transformations from the deployment level.

The CML library abstracts the low-level details contained in the IP-XACT library, which contains information about different concepts deployed in our approach (e.g. bus interfaces, component and design descriptions). Subsequently, the MARTE model created in the deployment phase is parsed to obtain an IP-XACT system description, which contains the component instantiations, their interconnections, and the configurable elements (CE). Depending on the type of CE, some might require to be set

at this stage, whereas others are to be configured by external tools in the flow. The way different IP-XACT concepts are used in our methodology will be further discussed in Section 4; the mapping between MARTE and IP-XACT objects can be found in previous work [13].

The IP-XACT descriptions for the top level design and for the constituent IPs are fed to so-called Platform configuration tool. We make use of Sodius MDWorkbench [14], a MDE platform which enables the definition of meta-models and to carry out model transformations.

The IP-XACT XML schemas have been processed by an improved XSD/Ecore meta-model importer in MDWorkbench, which leads to a Java/EMF implementation of the IP-XACT meta-model. The tool is in charge of producing the desired outputs for the rest of the flow, by using transformation rules that can be tailored to the design flow needs, or evolved depending in the back-end requirements. In this work, we have decided to target the Xilinx EDK design flow [15], as explained previously. This allows us to build SoC based platform which is easier to manage and in which DPR concepts can be integrated in the design flow.

Xilinx EDK makes use of a collection of files defined in the Platform Specification Format (PSF) document [16], which formalizes the description of different components in the Xilinx design flow for processor-based systems. Being able to handle such Xilinx platform models allows interoperating with platform-independent standards such as IP-XACT. Xilinx PSF files are structured in a textual format, which can easily be understandable by machines by defining a parser, but the first mandatory step if the meta-model definition. The Ecore formalization of these meta-models does not exist, and has to be entered, in UML for instance. We have created meta-models for the different Xilinx files used in the EDK environment, such as the Microprocessor Hardware Specification (MHS) and the Microprocessor Peripheral Definition (MPD), among others. As an example, Figure 2 depicts the UML Model for the MHS file, used to describe how a top-level is implemented in Xilinx EDK.

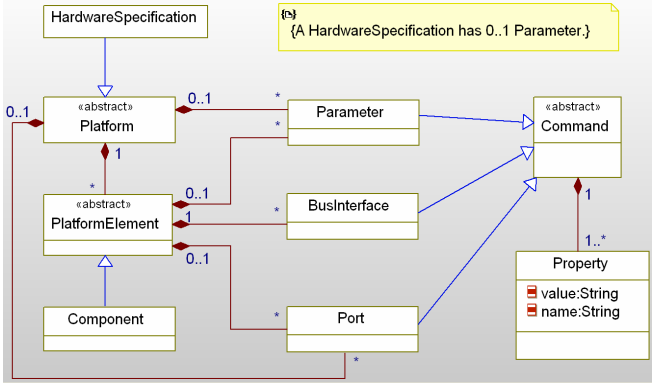

Fig 2. Model UML Model of Xilinx PSF Meta-model – MHS

3.2. Model transformation tool and intended back-end.

In our approach, Sodius MDWorkbench model tool provides a means of generating different outputs that are to be fed to the Xilinx EDK XFlow. As mentioned before, the Code Generation phase produces a set of files that are used by EDK to describe a SoC platform intended for implementation in Xilinx FPGAs. The MHS file is deployed to generate the top-level implementation of the embedded platform by using the PlatGen tool.

In this way, the top level VHDL implementation is obtained and can be synthesized; this intermediate architecture can be used for testing the platform, by using non-DPR implementation of the IPs. Individual IP specification is achieved through the use of files such the Microprocessor Peripheral Definition (MPD). Generating it for already existent IP cores (e.g. DDR2, TFT Display, RS-232) is redundant. However, being able to handle these files in MDWorkbench, enables for their parameterization.

Nonetheless, the Xilinx DPR flow requires, at the entry design phase, the definition of the top-level implementation and of the individual DPR IPs, which are intrinsically parametrisable. The features added to the IP in terms of their interfaces and functionalities affect the resulting Xilinx EDK specification files. Therefore, configuring such IPs at a higher level of abstraction facilitates the entry design phase. Therefore, by using IP-XACT, we provide a means to integrate the high-level models in a generation chain; IP-XACT functions as glue intermediate level, allowing us to represent the intermediate components and to obtain the necessary IP metadata for the MARTE deployment models. Subsequently, the IP-XACT desing description in converted to an EDK MHS description, used for creating the top-level HDL design that, along the IPs descriptions, is fed in the form of netlists

4. IP-XACT FOR COMPONENT DESCRIPTIONS.

In the previous section we have provided hints on how IP-XACT is used as an intermediate representation, but also as the backbone for passing from a MARTE description to an EDK system, via model transformations. In this section, we further describe how certain IP-XACT concepts are used in our methodology, emphasizing the link to concepts in MARTE and EDK.

The standard defines four central object descriptions, which are *bus* and *abstract definitions*, *component*, and *design* descriptions. These four elements are sufficient for structurally describing a system and the IP cores the compose it. We concentrate then our efforts in describing IP-XACT components, specifically *hierarchical modules*, for which a sub-system design description is attached.

A *component description* packages the information related to an IP core, as depicted in Figure 3. We have

chosen this block-like representation of the IP-XACT concepts instead of the schemas in the standard, since it facilitates their comprehension. The elements contained in the component schema are intended for describing as many different kinds of IP cores as possible, but it is obvious that not all of them will be required in all instances. Here, we have included the most widely used concepts for structural and logical implementation and parameterization. In the next sub-sections, we will briefly discuss how the different elements are exploited in our approach.

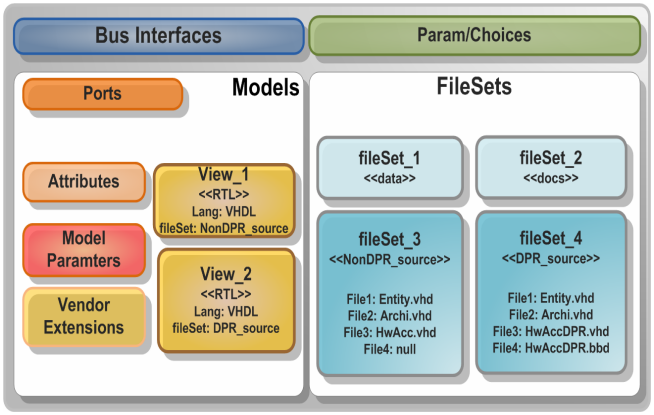

Fig 3. IP-XACT concepts for a component description.

4.1. Bus interfaces

Each IP core identifies one or more <*busInterfaces*>, which are groups of ports that belong to an identified <*busType*>. Apart from defining standard bus interfaces, they can be used for other ports of the IP, such as DMA or VFBC ports.

The ensemble of these interfaces comprise the ports in the VHDL entity, and in EDK, they appear in the MPD file to define the ports of the IP. In both cases, the information is used for IP integration and top-level stitching. Moreover, core ports external to the FPGA need to be defined in the User Constraints File (UCF).

4.2. Parameters and Choices

The <*Parameters/Choices*> sub-elements permit the configuration of an IP core, and enable to automate the parameterization of an IP within a tool flow. <*Parameters*> are normally set at the design level, and only if they are defined as being configurable in the IP-XACT description. The <*resolve*> attribute in the configurable element description defines exactly how a parameter must be set.

In our approach, most of the parameters are *immediate*, their values being obtained from the MARTE description itself. In other cases, certain values can be set by the *user* (e.g. through a GUI), be *dependent* on other parameters in the IP-XACT description (using a dependency equation that takes the parameters IDs as inputs), or *generated.*

4.3. FileSets, Models and Views.

The <*fileSets*> element contains a list of the <*files*> and directories associated with a component. These files might include drivers, implementation files, netlists, and other files related with a particular tool. Files can be grouped for describing particular functions and purposes, greatly promoting EDA and IP reuse. This separates the high-level models from the intended back-ends, which can adapted, thus providing extensibility mechanisms.

The <*model*> element describes the <*views*>, <*ports*> and model-related <*parameters*> of a component. An IP can contain different views such as RTL, TLM, software, and documentation, to name just a few. Views are used in tandem with <*filesSets*> and generators to enable the automation of component related tasks, such as FPGA synthesis and source code compilation.

In our methodology, we exploit this capability of the <*view*> *elements* for describing components with different purposes, but having the same interface. The Xilinx DPR design flow requires, as inputs, the netlists of the top-level design, but also of those DPR modules. In the latter case, the modules functionality must be synthesized independently, while maintaining the same interface in the top level implementation. Let us consider the example in Figure 4, which depicts the implementation of a simple IP attached to a bus via the Xilinx IPIF interface.

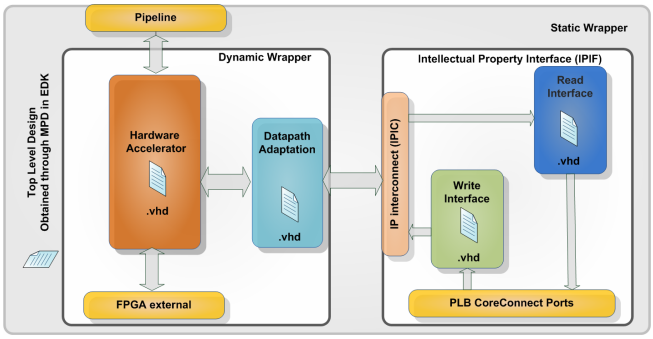

Fig 4. IP-XACT abstract representation for the DPR IP

Each of the depicted components corresponds to a VHDL file. The hardware accelerator (HWA) is wrapped by what we call a "Static Wrapper", which in our example is represented attachment to the IPIF interface. Additional components might be necessary for attaching a given HWA to the IPIF logic, for instance a Protocol Adaptation Logic (PAL). Along with the HWA, the PAL comprises the so-called "Dynamic Wrapper" (DWR).

The DWR corresponds to the logic that is to be swapped during the DPR run-time phase, whilst the rest of the system (and in this case, the static wrapper), remains unchanged. The dynamic wrapper corresponds to a black box (BB) that defines logically where the Partially Reconfigurable Modules (PRMs) are to be mapped.

Afterwards, this BB will be assigned to a fixed Partial Reconfigurable Region (PRRs) during the floorplanning phase of the DPR design flow. This means that, for the DPR design to be generated, the functionality of the DW must be synthesized independently.

The View_1 is associated to a <<NonDPR_source>> *fileSet*, which contains all the implementation files, including the DWR logic, whilst the View_2 references a *fileSet* called <<DPR_source>>, which makes reference to an IP-XACT Black Box element. Since the component instances and interconnections remain unchanged, only a design description, which is obtained from the MARTE model, is necessary.

5. CASE STUDY FOR SYSTEM IMPLEMENTATION.

In this section, we present a case study in which we show how the methodology is used to implement a MicroBlaze-based SoC platform integrating some DPR blocks. We start by describing MARTE related modeling concepts for the individual components and the platform, and how they are related to their IP-XACT counterparts. Afterwards, we describe how a DPR component can be parameterized and integrated from a MARTE description. As mentioned before, DPR cores require non-DPR and DPR descriptions using different views, to target different scenarios.

5.1. Parameterization View.

At the deployment level, the designer of the DPR application has enough information to choose the components to build its application; these components are gathered from a CML library, which contains abstract representations of the IPs, described in MARTE.

The designer creates first what we call a "Parameterization View", which contains the set of components to be used in the platform. Figure 5a) depicts a section of this view, showing three components: PLB_UART, PLB_HWICAP, and PLB_TFTcntr. Each component contains a specific MARTE stereotype; most of the components in this figure are <<HwComponent>>, since they reference real IP cores. The components in the CML library are related to their IP-XACT counterparts by their VLNV value (IP-XACT unique identifier for objects). Each class in this diagram holds references to the configurable elements that this component instance contains, and which must be set by the designer. This view has been separated to avoid cluttering the models. Thus, only those parameters with "immediate" attributes are visible in the "Parameterization View". Apart from having an immediate attribute, we have defied vendor extensions to specify dependencies and validity, in order to control which parameters, interfaces or ports can appear in the MARTE component.

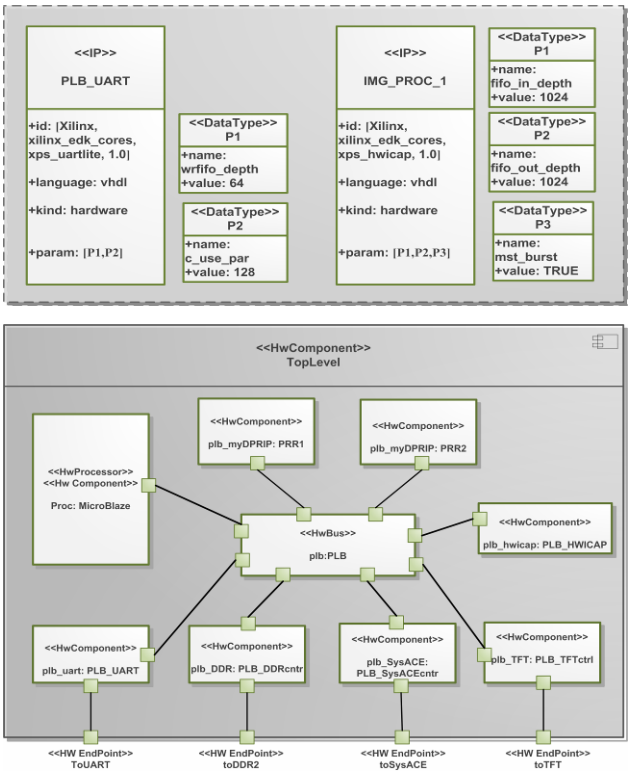

Fig 5: Modeling of the system: a) Parameterisation and b) Platform Views

5.2. Architectural View and Mapping to IP-XACT

On the other hand, Figure 5b) shows the modeling of a reconfigurable system, targeted to an embedded architecture to be exploited by the Xilinx's EDK environment. This diagram represents a merged functional/physical view of the system used to express the attributes related to physical/logical stereotypes. This is the so-called "Structural View"; using a CSD, the designer is interested in describing the way the system is to be connected, not concerned to the low level aspects of the design.
Every component instance has two type definitions, one being functional (the type of modules) and the other physical (e.g. areatype, Static or DynamicReconfigurable). In this case study, we make use of two dynamic reconfigurable regions (labeled as PRR in the diagram).

Both descriptions are parsed for generating the IP-XACT description for the top-level implementation. As mentioned before, the "Structural View" is used for generating the IP-XACT design description of the top level design and of the hierarchical IPs.

In order to perform the aforementioned mapping, a step must be defined in which the MARTE specification is parsed. Certain elements in the MARTE platform model will correspond to IP-XACT objects in the model library. The objective is to identify all the elements in the platform specification, generating a top-level design file. Table 1 defines these mapping for obtaining the IP-XACT design.

Table 1. Transformations from a deployed model to an IP-XACT design.

MARTE	IP - XACT
CSD Diagram	**Design**
Part	spirit:ComponentInstance
id: identifier	spitir:ComponentRef
Parameter = value	spirit:configurableElement:ReferenceID
Connector = name	spirit: interconnection portRef (busIF name)
Ednpoint = name	AdHoc Connection = name with intPortRef and extPortRef

These transformations have been implemented using Sodius MDWorkbench, where our Xilinx XPS and IP-XACT meta-models reside. The obtained IP-XACT design is fed to our tool, where is used to generate the files used by Xilinx to obtain the synthesizable top-level description.

5.3. Modeling of the DPR Cores.

We have chosen to concentrate our efforts in targeting EDK compatible IP cores, since we aim to generate DPR systems in a Xilinx flow. Figure 6 shows a detailed block representation of the PRM components in Figure 4. We have implemented an image processing IP that receives a complete line of the image and stores it in FIFO_in; when a complete line has been stored, is processed by the HW accelerator and the results written to the FIFO_out component.

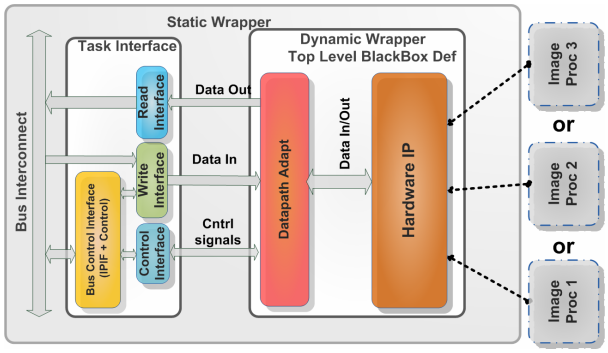

Fig 6.) Underlying HDL description

The resulting processed line is sent via a master to the DDR memory. The two FIFO components and their control logic are the PAL; this component is associated with a VHDL file describing its functionalities. The HWA is associated with another VHDL file. Similarly, the IPIF logic containing the write/read interfaces is represented by another VHDL file, which is generated by choosing the required services through a GUI. If the PAL changes between changes of the IP core in a DPR system, forms part the DPR Wrapper, this is, the Black Box used by the Xilinx DPR flow to specify that a component in the design is to be a PRM. This is the case of our example. The rest of the IP is labelled here as Static Wrapper, since its functionalities do no change between different configurations of the IP.

We have modelled the DPR IPs in our approach in such a way that, by using the <*views*> and <*fileSets*> elements of the IP-XACT description we provide a means for pointing the location of the different implementations of the DWr into the IP implementation directory. The <*fileSets*> element in the component description specifies all the files used to describe a component. In particular, a least one <*fileSet*> is destined for specifying the HDL sources using to implement the IP functionalities. A component can contain multiple implementations, each represented by a <*View*> referencing a <*fileSet*>, as depicted in the IP-XACT component description of Figure 3.

The <<DPR_Source>> fileSet, which is parsed during the generation phase to retrieve the location of the different implementations of the IP, which are synthesized separately, as required by the DPR design flow, and that otherwise has to be done manually and using a separate tool (Xilinx ISE). In Figure 6 we show a block representation of the implemented IPs; there are two of them in the platform of Figure 5b): each of them treats a half of a input image and sent to the TFT controller in the card for display, and implemented several image processing tasks (binarization, inversion, edge detection, and greyscale reduction) was hardware accelerators.

Since we are targeting the Xilinx EDK design flow, we have decided to implement all our IPs following the directory structure depicted in Figure 7, which is used to separate HDL implementation files from others used by the Xilinx tools; the function of the different files is out of the scope of this paper, for a more information, the reader is director to the Xilinx platform specification guide [16].

The most important aspect, as previously discussed, is the location of the implementation files that lies under the HDL directory. The IP-XACT component description contains this information under the <*fileSets*> element; when choosing a component in the top-level MARTE description each of the possible implementations is referenced via the <*View*> elements, pointing at the location of the corresponding HW accelerator implementation.

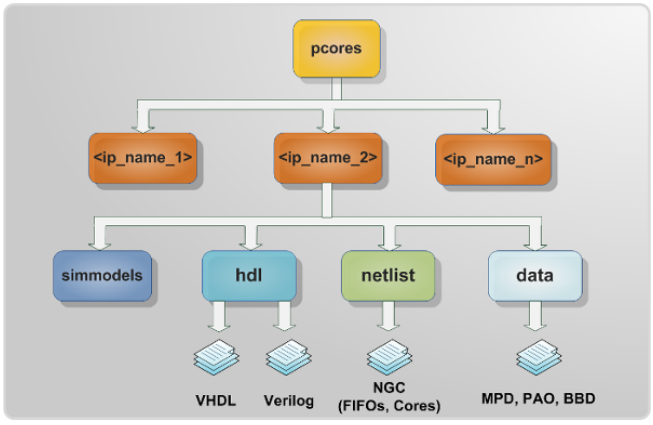

Fig 7.) Underlying HDL description

6. RESULTS AND DISCUSSION.

The system described in Figure 5 has been used for implementing an image processing platform. In Table 2a) we can observe the implementation results of two simple pixel-based operations, as detailed in the previous section. The resource utilization of each of the PRMs in the same; we provide as well the partial bitstream size (5KB for each), which means that for using the throughput provided by the HWICAP we can attain a configuration time of 50 microseconds. In Table 2b) we provide a more complex example, in which we have mapped a Discrete Cosine Transform (DCT) and the Discrete Wavelength Transform (DWT); as it can be observed, the increased resource utilization increases the partial bitstream sizes and accordingly, the configuration times.

Table 2. Implementation details for two image processing implementations.

a) Example 1	Resources Utilization					
DPR Module	LUT	FF	DSP	RAM	KB Size	Conf Time
IMG Proc 1	1008	847	0	0	5 KB	0.05 ms
IMG Proc 2	1008	847	0	0	5 KB	0.05 ms
b) Example 2	Resources Utilization					
DPR Module	LUT	FF	DSP	RAM	KB Size	Conf Time
DCT	1419	1636	8	8	47 KB	0.47 ms
DWT	940	389	0	4	44KB	0.44 ms

By combining MARTE and IP-XACT, we provide a means to facilitate IP deployment. However, the design effort is also significantly reduced; ff the system had to be created using a pure VHDL description, the design capture might take days, in process very prone to errors and difficult to maintain. Using EDK simplifies the task, but requires a great deal of expertise on the necessary tools and the used files if an MDE approach is to be adopted. Using our approach, which is based on the IP-XACT standard, facilitates the creation of EDK systems, since we provide a means to pass from MARTE to the Xilinx Platform studio formalism; not only the design effort is reduced (as depicted in Table 3), but the maintainability is improved. Also, we can adapt the methodology to changes in the back-end by only modifying the transformations from IP-XACT.

Table 3. Design efforts using VHDL, XPS and the proposed methodology

Type of design capture	Time	Description
Pure VHDL Approach Manually integrating the platform	Days	Less reliable, long and prone to errors.
Using Xilinx EDK Platform Integration in XPS	1h30 min	EDK is justifiable for systems containing at least one processor (DPR manager)
Proposed Approach Platform Integration	40 mins	The time required for a platform creation is reduced, and the maintainability is improved

7. CONCLUSIONS.

In this paper we have presented a design methodology that enables the parameterization and integration of IPs into a DPR platform at multiple levels of abstraction. The presented approach uses the IP-XACT standard as a centric representation of the platform, and as backbone for federating the heterogeneous data used in the design flow of DPR based SoCs. Furthermore, as we have show in this paper, IP-XACT can also be exploited as a means for providing an intermediate system description to pass from MARTE models to an EDK back-end, our chosen target for creating DPR, processor-based systems.

Moreover, we have showed how IP-XACT can be exploited as a means to perform the parameterization of IPs and their subsequent integration into a design description. Our approach also promotes IP reuse in by integrating several implementations of the same IP, intended for different scenarios and whose selection can be selected from the MARTE description.

8. ACKNOWLEDGMENTS

This work has been supported by the ANR FAMOUS Project (ANR-09-SEGI-003) by the Agence Nationale de la Recherche.

The authors also wish to thanks Sodius for their support in this project and for providing access to their design tools.

9. REFERENCES

[1] P. Manet, “An Evaluation of Dynamic Partial Reconfiguration for Signal and Image Processing in Professional Electronics Applications”, EUSASIP Journal of Embedded Systems, 2008.

[2]. Xilinx Corporation, Partial Reconfiguration User Guide, Xilinx UG208, 2011.

[3] S. Taha, A. Radermacher, S.Gérard, J-L. Dekeyser: MARTE: UML-based Hardware Design from Modelling to Simulation. FDL 2007.

[4] OMG, “Modeling and Analysis of Real-time Embedded Systems, MARTE), 2009.

[5] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows," IEEE Std 1685-2009, Feb. 18 2010.

[6] J. Vidal, F. de Lamotte, G. Gogniat, J.- P. Diguet, and P. Soulard, "IP reuse in an MDA MPSoPC co-design approach," Microelectronics (ICM), 2009 International Conference on , vol., no., pp.256-259, 19-22 Dec. 2009.

[7] I. R. Quadri, S. Meftali, and J.-L. Dekeyser, "Designing dynamically reconfigurable SoCs: From UML MARTE models to automatic code generation," Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on, vol., no., pp.68-75, 26-28 Oct. 2010.

[8] W. Kruijtzer et al., “Industrial IP Integration Flows based on IP-XACT Standards,” in DATE’08, March 2008, pp. 32–37.

[9] C. Lennard, “Industrially Proving the SPIRIT Consortium Specifications for Design Chain Integration,” in DATE’06, March 2006, pp. 1–6.

[10] C. André, F. Mallet, A. M. Khan, and R. de Simone, "Modeling SPIRIT IP-XACT with UML MARTE", In: Proc. DATE Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile, 2008.

[11] F, Herrera, E. Villar, "A framework for the generation from UML/MARTE models of IPXACT HW platform descriptions for multi-level performance estimation," Specification and Design Languages (FDL), 2011 Forum on , vol., no., pp.1-8, 13-15

[12]. G. Ochoa-Ruiz, E.B. Bourennane, H. Rabah, O. Labbani, “High-Level Modeling and Automatic Generation of Dynamically Reconfigurable Systems,” in Proceedings of the DASIP Conference, Tampere Finland. November 2011

[13]. G. Ochoa-Ruiz, O. Labbani, E.B. Bourennane, P. Soulard, “Model-Driven approach for automatic dynamic partially reconfigurable ip customization, RAW 2012.

[14]. Sodius Corporation, MDWorkbench, http://www.mdworkbench.com/, 2011.

[15]. Xilinx Corporation, Embedded System Tools Reference Guide, Xilinx UG111.

[16] Xilinx Corporation, “Platform Specification Format Reference Manual”, UG642